SENSOR FOR DETECTING AND ELIMINATING INTER-PROCESS MEMORY BREACHES IN MULTITASKING OPERATING SYSTEMS

FIELD OF THE INVENTION

[0001] The present invention relates to the field of protecting and securing data in computerized systems. More particularly, the invention provides a method and system for detecting inter-process memory breaches in multitasking operating systems.

BACKGROUND OF THE INVENTION

[0002] Modern operating systems are actually a modular collection of building blocks rather than one monolithic object. This form of architecture enables an Operation System (OS) manufacturer to build and distribute new facilities (or new versions for existing facilities) with relative ease. It also enables third party programmers to add new capabilities to a basic operating system without accessing its source code, by means of well-defined extension interfaces.

[0003] In particular, the Input/Output (I/O) architecture of a conventional operating system is multi-layered, scaleable and extensible, i.e., each packet of I/O data travels along a chain of layers, wherein the layers are organized so that applications that are being executed by the OS at the system of a user are divided into separate functional components that interact in some sequential and hierarchical way, with each layer in the chain usually having an interface only to the layer above it and the layer below it. Some of the layers are those provided with the original software package of the OS, some are updates, others are additional layers not present in the original software package, and finally, some are actually built by third party suppliers (including original parts of the original software package that were subcontracted).

[0004] Operating systems usually make a distinction between a privileged mode and non-privileged mode regarding the ability of a process to call 'privileged' services. The notion of multi-layered architecture and underlying extension mechanisms apply to both modes, but the implementation may differ significantly.

[0005] A Privileged mode (so-called "Kernel-mode") is the essential core of any OS, which provides basic services for other parts of the OS. Typically, the Kernel-mode is the part of the OS that resides in the memory of the computer at all times during its operation, and provides basic services. It is the part of the OS which is closest to the machine level and may directly activate the hardware of such a computerized system, or interface with another software layer which drives a hardware. Due to performance considerations, kernel-mode processes typically share the system's physical memory space without an extra mapping of their non-privileged mode relatives. A kernel-mode process can be seen as a server to many non-privileged mode processes, which is vulnerable to possible low-level breaches. A process of a non-privileged mode (so-called "user-mode") can call system services that are not privileged.

[0006] In particular, user-mode multi-layer extension mechanisms are quite vulnerable to memory-space breaches. These enabling mechanisms are relatively well documented, and a dedicated programmer having access to the interfaces of these extension mechanisms is generally able to implement them in quite a short time. Some well known books in the field of general Operating Systems, particularly Windows™ OS provide relevant information regarding this subject.

[0007] From now on, and unless otherwise stated, the following text will refer to user-mode.

[0008] The building blocks of standard applications, such as a word processor, an Internet browser etc., are code modules, usually divided into program modules and/or into Shared Code Resources (SCRs). Examples for such SCRs are the Dynamic Link Library (DLL), which are included in the Windows OS of Microsoft. Furthermore, each application may use several SCRs on the same session. Generally, SCRs are grouped in stacks, wherein each stack contains several SCRs, sometimes a dozen or more SCRs are grouped together in one stack. The SCRs are organized in each stack in a chain-like manner. Normally, whenever a service from a specific SCR is requested by a user application, the request travels along the whole relevant stack, however, the user application has no clue about the specific SCRs that actually serve it along the way.

[0009] Moreover, when an extension is needed to one of the OS services, for example, encrypting certain I/O data packets, an insertion of an SCR into the relevant stack chain should do. Of course that SCR has to comply with a given interface and be good mannered, the least it should do is to dispatch incoming calls to the next SCR in the chain.

[0010] In a typical case, an extender, which might be, for example, a specific process, requests from the OS to insert an SCR into a specific extensible chain. If all goes well, the SCR is inserted as a new "layer", and starts receiving relevant calls as if it was an original part of the stack, and of the OS. From this point on, until this new SCR is appropriately removed from the chain, the newly installed SCR is mapped into the address space of any application that happen to use that relevant stack.

[0011] Due to resource-economy considerations, a reasonable multitasking operating system would load just a single copy of a given SCR into the physical memory, and then map it to the virtual address space of each process that might need it. More particularly, each 'instance' of the SCR is mapped to the appropriate process context. Unfortunately, there is more than one way to share memory between the SCR's 'instances'.

[0012] Combining the aforementioned factors that compromise the requirement for separation between memory spaces of different processes, there is an opportunity for offenders to abuse the inherent mechanisms of the operating system. In fact, this provides a possible way for one process to break into the memory space of another process.

[0013] An offender that has managed to break into the memory space of another process has a choice of options. Amongst other threats, the offender may read or manipulate I/O, it might change the behavior of the invaded application, or it may send information from one process to another process.

[0014] One of the most serious aspects of memory-space breaches is the ability of the offender to take the identity of the invaded process. This makes life harder for auditing